FINESSE

John Henry Smith

United States
Naval Postgraduate School

THESIS

FINESSE

by

John Henry Smith

June 1970

Finesse

by

John Henry Smith
Lieutenant Colonel, United States Marine Corps
B.S., United States Naval Academy, 1953

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

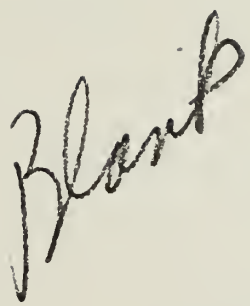
from the

NAVAL POSTGRADUATE SCHOOL
June 1970

ABSTRACT

Finesse is a program written to investigate the role of a generalization scheme in computer learning. The finessing play in the game of bridge is developed as an example in which the computer starts with a few basic bridge rules and improves its play through self-analysis of successive plays of the same hands. After deciding that an optimum play has been found for a particular hand, Finesse makes the generalization that all hands with approximately the same distribution should be played in the same way.

TABLE OF CONTENTS


Blank

I. INTRODUCTION

This paper describes an interactive bridge-playing program, Finesse, designed to solve finessing problems in no-trump hands. The program is written is PL/1 in two versions, one for batch processing, the other to be played on-line. The object of the thesis is to investigate computer learning using Finesse as the vehicle. This was done by writing a program which produces its own set of finessing rules; rules which continue to build and improve as the program plays more and more hands. As new and different card combinations are encountered, the program amends the rules stored in memory either by adding to the knowledge contained, or when that knowledge proves faulty, deleting from memory those rules which no longer apply.

Finesse was designed with the idea that it should play in as humanistic a way as possible. A beginner in any game starts by learning a few basic rules, then builds on these as his degree of expertise increases. New rules are generally learned as new situations are encountered. This is the approach taken in Finesse. A few simple rules were built into the program so that Finesse would have a basis for making an initial play for any given hand. These rules were designed to insure that players follow suit whenever possible and that certain rules of distribution are followed in the initial play of a hand. For instance, if declarer's hands contain eleven or more cards of the same suit, there

would be little requirement to finesse for the King. Another example of this is the popular "eight ever, nine never" rule by which beginners learn to finesse for a missing Queen if they hold eight or less cards, but to play for the Queen to fall if nine cards are held. Such heuristics are refined and amended as the analysis of play directs.

Before any programming could be done, certain basic decisions had to be made concerning the scope of the thesis and the direction that its development would follow. It quickly became obvious that time would not permit the writing of a program to play all aspects of the game and, more important, that such a program would not be required to study the stated problem of computer learning. For the finessing problem, the playing of one suit with the intent of winning the greatest number of tricks has enough of the same strategic and tactical problems found in playing the entire hand, that it is a worthy vehicle for the purpose of this thesis. In short, finessing can be considered bridge in miniature.

Certain restrictions were placed on the type of hands to be played in Finesse, thus reducing programming problems without detracting from the thesis objective. For example, with the use of no-trump hands only, the problem of counting and keeping track of trump cards was eliminated. Also, certain information that would normally be available and useful, such as the bidding sequence, is unknown in Finesse. Omitting this type of information provides a more generalized line of play than might otherwise be found.

II. BACKGROUND

Several authors have published papers on bridge programs, some of which consider the play of all aspects of the game. Until 1969, however, no reference could be found to show that any real efforts had been made toward generalizing on the results of completed plays and establishing a line of play for similar hands that follow.

The earliest three papers on bridge-playing programs were published in 1962. Two of these were the result of individual thesis work at Massachusetts Institute of Technology, by E. R. Berlekamp and Gay Carley (Refs. 1 and 2). The third, by Thomas A. Throop (Ref. 3), is a short description of a program to conduct bidding for one round. The programs written by Berlekamp and Carley were more ambitious and deserve a closer look.

Carley's thesis describes a full-length bridge program written in assembly language for the IBM 7090 computer. The program bids, decides upon a strategy for the hand, and plays a fair game of bridge, largely through programmed heruistics. The program is flexible in the sense that it contains a number of subroutines which could be easily changed by the author. But these changes were not controlled by the program as the result of any self-analysis of previous plays, and this should be the object of a generalization process.

In the case of Berlekamp's paper, a generalization process is included. Here the author has constructed a program to solve the specific

bridge problem of finding the winning play in seven no-trump hands in double-dummy bridge. The final solution for a particular hand was reached by first playing it in accordance with a set of heuristics built into the program. Then the same hand was re-examined with a different defensive play. If the original line of play was again successful, a new defensive strategy would be tried until either all strategies had been played, or a particular defense proved better than the previous plays. If this latter event occurred, a new offensive play would be tried and the play and test procedure would be repeated. In this manner, the play of a particular hand would improve until the optimum play had been found. Thus, this program generalizes only on the line of play to be followed with one specific hand when all tricks must be taken.

Riley and Throop (Ref. 4) presented their paper in 1969. It describes what appears to be the first bridge program to attack the problem of generalization, including generalizing in finessing. Riley and Throop's approach to the finessing problem was to handle it through pattern analysis, with each finessing hand falling into one of eight patterns. This method differs from the method used in Finesse, which makes generalizations according to previous plays and stores them in decision trees. New and better plays are found, primarily through a trial and error process, much as they were in Berlekamp's program. The inferences drawn and rules developed are not made for one specific hand, but for any hand in which certain key cards are missing.

III. RATIONALE

In Finesse, the declarer's hands (North/South) are played by the computer, while a human opponent plays the East/West cards. The program learns to play the cards in much the same manner as a human player does. It starts with the knowledge that there are certain basic rules which must be followed:

1. All players must follow suit whenever possible.

2. If a player cannot follow suit, a card in another suit, preferably a loser, must be played.

3. When specific numbers and denominations of cards are held by the declarer and his partner, they should be played in a prearranged manner to win the most tricks. For example, if nine or more cards are held by declarer and the Queen is the only missing honor, the probability of it falling if the Ace and King are played out is greater than 50%. This play would be dictated if no further information were known concerning the opponents' distribution. With fewer than nine cards, the probability of the drop falls below the 50% level and a finessing play is required. Similar rules require Finesse to play for the drop of the King when more than ten cards are held and for the drop of either the Jack or Ten when more than six cards are held.

4. No suit with fewer than seven cards in declarer's hands has been considered as a finessing suit in this program. This is not because such finesses cannot be made, but because suits in which declarer has

fewer cards than his opponents are not normally identified as advantageous for early development in the play of a No-Trump hand. These suits are usually held in reserve for transportation purposes, stoppers in short suits, and for sure tricks when needed either to make the contract or avert a serious loss.

5. For a similar reason, no suit distributions with more than two missing honors have been considered. Had such consideration been given, a small extension of the program would have been required to determine the initial strategy for playing such suits, but the actual play of the hand would be similar.

6. If the suit to be played falls into the category of a finessing suit, then the missing honor(s) is singled out and designated as the card(s) to be located. The idea of identifying the cards declarer is missing, rather than those he has, was decided upon to cut down on the number of possible card holdings which could arise.

If a suit is identified as one to be played for a finesse, the next lower card to the missing honor, or a card of equivalent value, is selected as the finessing card and the initial finesse is made. The program will determine whether North or South must be the leader in order to finesse properly and will use transportation cards to position itself in the right hand if necessary. If the finesse fails and the missing honor takes the trick, Finesse will assume that there are no more key cards left in the suit to be concerned with. When the declarer has the lead again, the program will play out all remaining cards, anticipating the capture of all

unplayed tricks.

If the finesse is successful and the missing card does not appear, Finesse will assume that it has located the missing honor. It proceeds to make as many finessing plays as are required and/or possible through the opponent holding the missing card. When this card appears, Finesse will infer that all remaining cards should be played out. If at any point in this play an opponent wins a trick, the declarer's strategy does not change. Upon regaining the lead, he continues to draw the remaining cards.

As each trick is played, a tally is kept of the number of tricks won by declarer, and the identity of all winning opponent cards is stored in memory. When all cards in the suit have been played, an analysis is made of the number of tricks won by declarer as opposed to the maximum number he could have won. If, for example, declarer's cards were split 4-3, he could have won at most four tricks. If he were also missing the Ace, he could have won only three. In this case, if he actually won three, Finesse infers that the hand was played correctly and the play terminates, indicating that the optimum play had occurred. If this cannot be determined, then the suit must be played again, using a different strategy, and the results of the two plays will be compared.

The strategy to be followed in the second play is determined by the highest ranking card that won a trick for the opponents. This becomes the new card for which the program finesses. After it appears, the finesse is made for the original missing honor. The remaining

cards are then drawn as before, the tricks are tallied, and another comparison is made. If fewer tricks were made on the second play, the play is discarded and the original play is assumed to be the best. Otherwise, the new play is stored and will be used again the next time that combination of missing cards is encountered. This process of searching for a better play can be continued until the play analysis shows that there are no more alternatives, or that the last play is inferior to its predecessor. In this way the "optimum" play is left in memory until the same situation is encountered again and a better play developed.

IV. PROGRAM EXAMPLES

To see how Finesse works, let us study the plays of two different hands. The first has a suit distribution as shown below:

North

K J 5 2

West	East
10 8 4 3	Q 7

South

A 9 6

Finesse first analyzes the hand and determines if at least seven cards are held in the North/South hands. If so, it looks for the highest ranking missing honor. In this hand the Queen is recognized as this key missing card. Since the declarer only holds seven cards, a finesse is required. North, having been predetermined to be the leader, leads the Jack to finesse for the Queen. East, as played by the human player, covers an honor with an honor. South rises with his Ace to take the trick and West plays low. The program decides that North/South is the winner and signals that they have one trick. Since the trick was taken by South, the lead shifts to that hand.

Since the key card has been played, Finesse plays the remaining cards in descending order. The six is led by South to North's King, with West playing low and East giving up the seven. North then leads

his low card, the two, back to the nine, the highest card left in declarer's hand. When East shows out, West wins the last two tricks with the ten and eight. The values of these two cards are stored, as is declarer's final trick count of two.

Finesse then determines that the ten was the highest ranking of the opponents' cards to take a trick and designates the ten and the Queen as key cards. The hand is played again with the finesse for the lower of these two key cards, the ten, being tried first. Once again North is the initial leader. This time North leads his two. When East plays low, North attempts a finesse with the nine. When West's ten wins again, it is stored in memory and the play continues.

Since the ten has been played, the Queen again becomes the critical card and the Jack is led. At this stage the play follows the original hand with the Queen falling to South's Ace and West playing low. The remaining cards are played out with South leading the six to the King, West playing the four, and East showing out. Again, declarer wins only two tricks as West's eight takes the last trick. Analysis of the hand shows no reduction in the number of tricks won over the previous play. Finesse, applying an heuristic, assumes that the second play, attempting a deeper finesse, is a superior play to the first.

The evaluation after this second play shows that the eight is the new key card and adds it to memory. With North leading again, his two is covered by the seven and South's nine loses to the ten. Then the five is led and South's Ace takes East's Queen with West playing low.

South's last card, the six, is then led and West plays the four which is taken by the Jack as East shows out. North's last card, the King, is then led and West's eight falls to it, winning three tricks for declarer. Analysis of the play shows that one more trick was taken. Since no new key cards were discovered, Finesse infers that the optimum play has been found. The analysis is concluded, all key cards are left in the decision tree, and play terminates.

In a second example, the suit distribution is as shown:

North
10 8 3

West
J 9 5

East
K 7 4

South
A Q 6 2

With North in the lead, the initial play of the suit is as follows:

Play #1: King recognized to be missing.

Leader	Lead	Opponent 1	Leader's Partner	Opponent 2	Trick Count
North	3	4	Q	5	1
North*	10	K	A	9	1
South	2	J	8	7	0
South	6	out	out	out	1/3

*Lead changed hands through transportation cards to take advantage of the successful finesse.

Play #2: King, Jack recognized to be key cards.

Leader	Lead	Opponent 1	Leader's Partner	Opponent 2	Trick Count
North	10	K	A	5	1
North**	8	4	2	9	0
North	3	7	Q	J	1
South	6	out	out	out	$\frac{1}{3}$

**Lead changed hands to take the only finesse left.

Play #3: King, Jack, 9 recognized to be key cards.

North	8	4	2	9	0
North	10	K	A	5	1
South	Q	J	3	7	1
South	6	out	out	out	$\frac{1}{3}$

Three tricks were made in each play. After the final play, when no new key cards were found, play of the hand terminated. The results of the third attempt were stored as the program inferred that the deepest finesse is the best play.

V. MEMORY STRUCTURE

Finesse uses tree structures for storage of key cards and their associated finessing cards. Three different types of cells are used, and examples of each are shown in Figures 1 and 2.

The MCARDTREE cell is used only as a header cell for each decision tree. Each of these, along with its associated FPLAY cell, is initialized in the early stages of the program. The header cell contains the value of the highest missing honor in declarer's hands while the FPLAY cell holds the value of the finessing card to be used. The other fields of the header are pointers. M6 connects the header cell to its FPLAY cell and M1-M4 are initialized as null pointers. As the trees grow, these latter four pointers are used to connect the header cell to its descendents. The cells to be joined are those declared as N3 cells having a field holding the value of a key card and three pointer fields.

Four separate decision trees are used in Finesse; one for each of the top four honors. No Ten Tree is used, for unless the opponents' distribution is highly unusual (either 5-1 or 6-0), the ten will fall to successive plays of the other honors. If the unusual distribution should occur, Finesse will recognize it and play the hand accordingly without further recourse to a decision tree.

Since the header cell holds the value of the highest ranking honor missing from declarer's hands, any cards subsequently added to the tree must have a lower rank. The decision as to which pointer should be

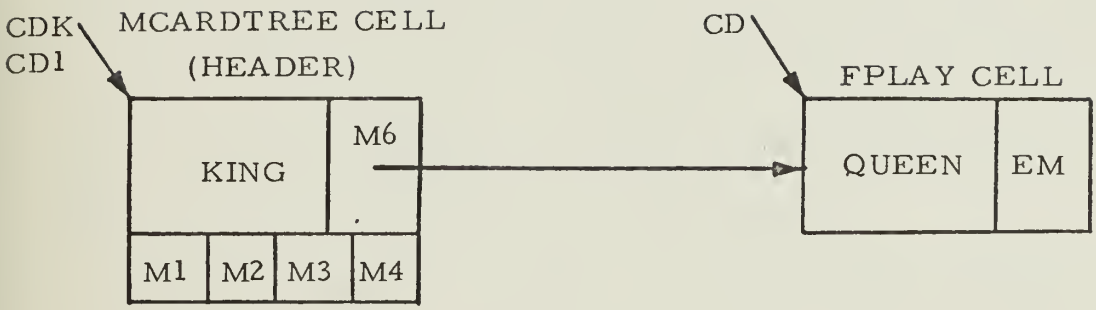


FIGURE 1

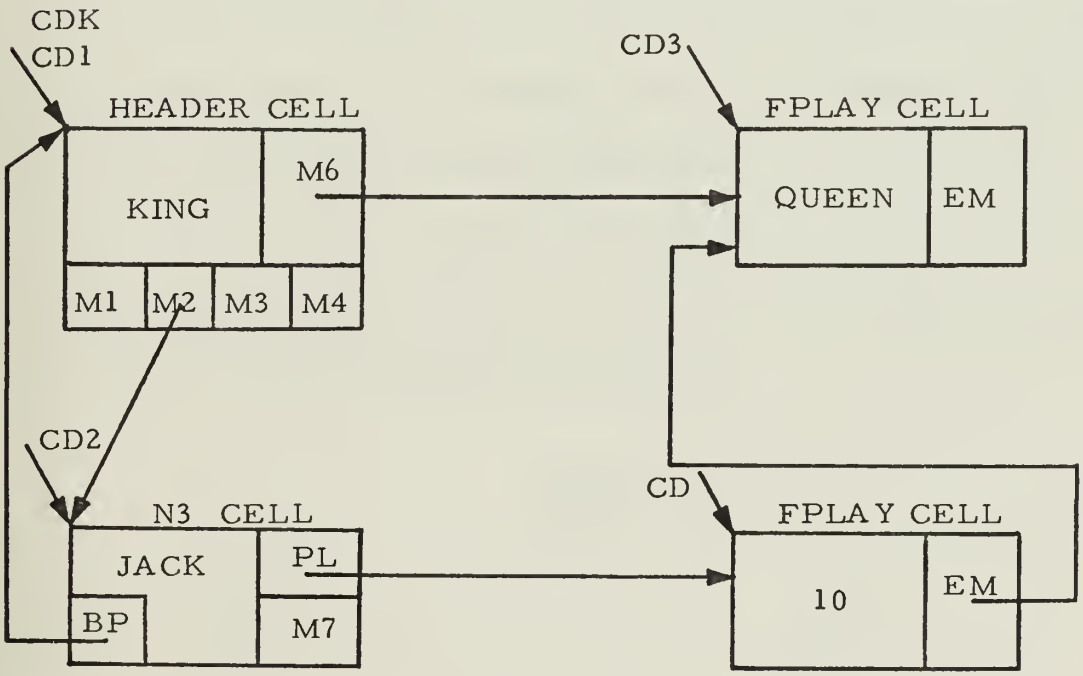


FIGURE 2

used to join these two cells together is made by comparing the value of the two cards concerned. If the card to be added is an honor and has a value of one greater than that of the card in the header cell, pointer M1 is used to connect the cells. If two greater, M2 is used, and so forth. If the card is not an honor, it is automatically pointed to by M1. Clearly then, only the Ace Tree will use all four pointers.

Like the header cell, each N3 cell has an FPLAY cell associated with it and containing the value of the card to be used in finessing. Pointer PL connects these cells. The other two pointer fields of this cell are BP and M7. BP is a back pointer used in moving up the tree when changes in tree position are made, while M7 points to the next lower N3 cell in the tree. The final pointer to be mentioned, the EM field of the FPLAY cell, is also a back pointer connecting the previous FPLAY cell in the tree. To better understand the tree construction, let us follow through with the addition of one level to the header cell, using one of the hands played earlier as an example.

North
10 8 3

West
J 9 5

East
K 7 4

South
A Q 6 2

Pointer CD was set to the header cell originally and CD1 was then set equal to it. A new N3 cell is allocated and a determination is made as to which of the M1-M4 pointers should be used. In this instance, with

the King in the header cell and the Jack the next key card missing, pointer M2 is chosen and set to the N3 cell (Figure 2). Level one pointers are set as shown and CD is moved to the N3 cell. BP is set to point the way back up the tree and a new FPLAY cell is allocated. All level two pointers are set, with M7 having a null value, and the construction is complete.

Subsequent levels are added in a similar manner with the setting of the M7 pointer to the new N3 cell rather than one of the M1-M4 pointers. CD continues to be used as the first pointer to be moved as each new cell is allocated and the other moving pointers follow it. Pointers CD1 and CD3 remain at the header cell level.

A level of cells may be moved easily up or down the tree by reconnecting the appropriate M7, BP and EM pointers, and rearranging the movable pointers so that CD, CD2, CD4 and CD5 always point to the lowest and next to lowest level cells. Figure 3 shows the decision tree after the final play has been completed.

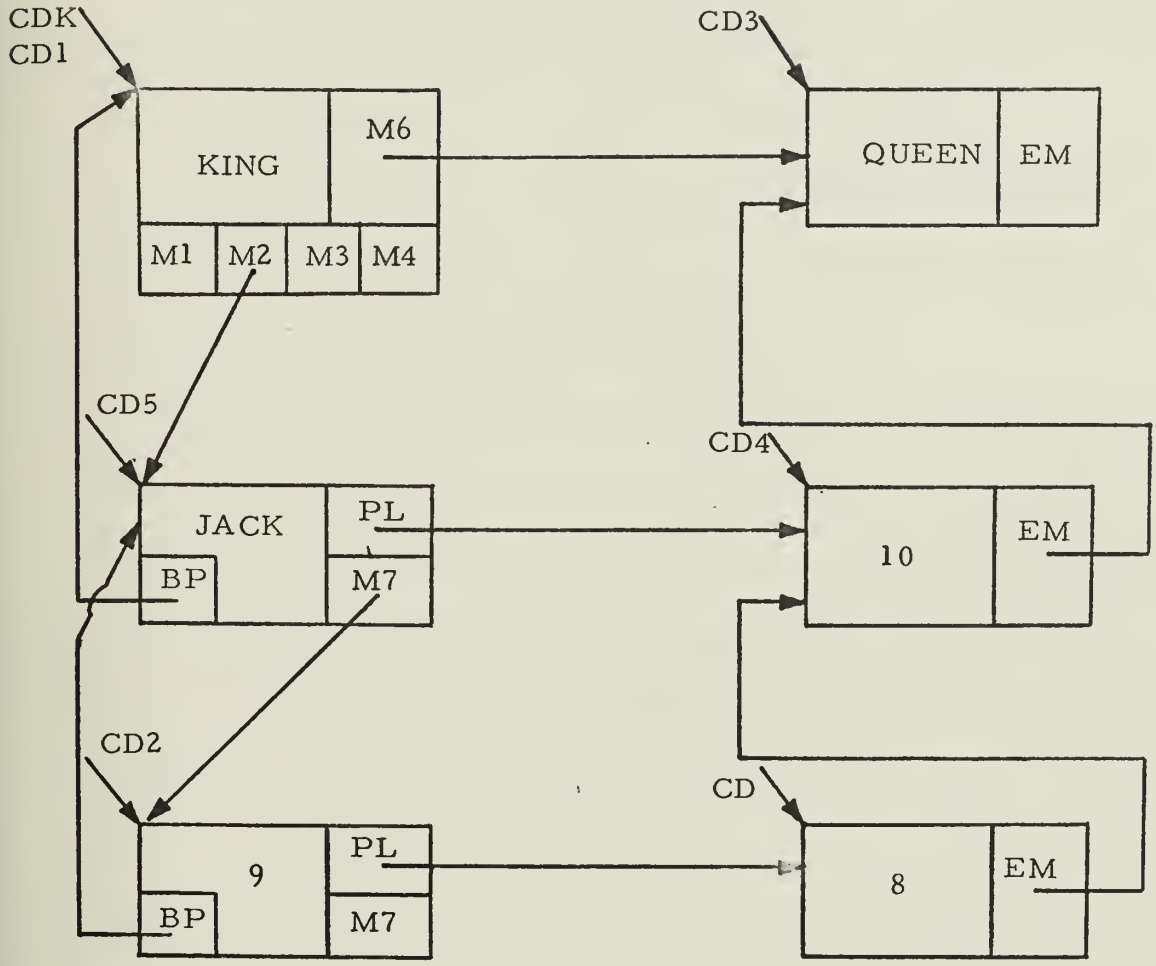

CDK
CD1
KING
M6
M1
M2
M3
M4
CD3
QUEEN
EM
CD5
JACK
PL
BP
M7
CD4
10
EM
CD2
9
PL
BP
M7
CD
8
EM

FIGURE 3

VI. PROGRAM DESCRIPTION

Finesse begins each new play of a suit by determining whether initialization of the decision trees is necessary. If initialization is required, the header cells and their associated FPLAY cells are constructed. Otherwise, this part of the program is bypassed so that the hands to be played may be read in. Figure 4, a Macro Flow Chart, shows how the play of a hand flows through the program.

Each player's cards are represented in an array consisting of thirteen bits, with a one indicating the presence of a card in the hand. The cards are represented in inverse order, with the Ace occupying the first array position, the King the second, and the two the thirteenth. Thus the highest ranking cards have the lowest array numbers. This simplifies the many searches required to locate the high cards of arrays HAND1 and HAND2. For input/output purposes, however, the card values are inverted and represented more normally with the Ace having a value of 14, King 13, Queen 12, Jack 11 and ten through two the values of 10-2.

All four hands of the suit to be played are read into the program. Either North (NH) or South (SH) is declared to be the initial leader (LEADER) in the play. The array HAND1 is set equal to the leader's hand, HAND2 equal to his partner's, and OHAND1 and OHAND2 setequal to the correct East/West hands. These arrays are then used as the playing hands and the North, South, East and West arrays are stored,

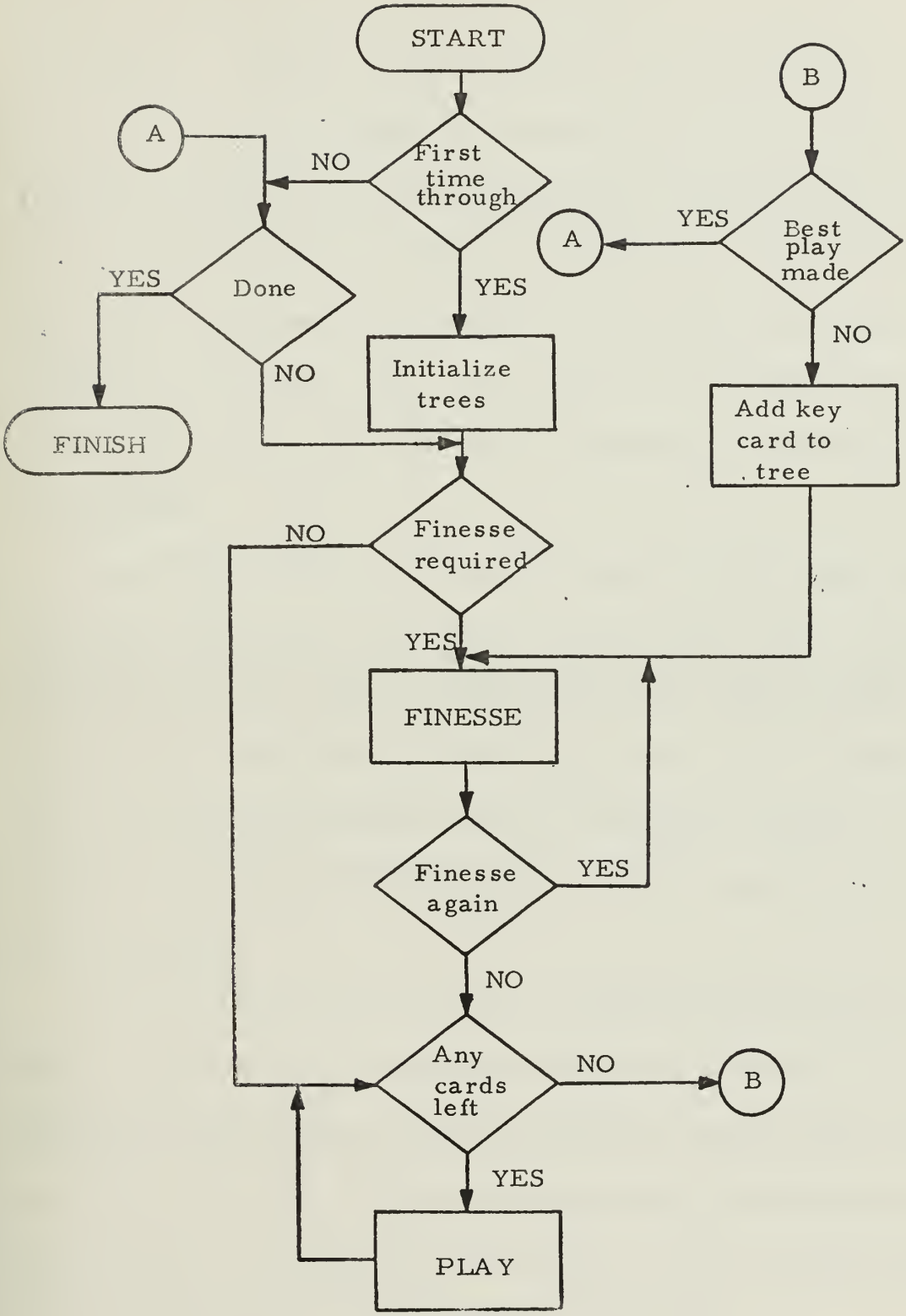

START
A
First time through
NO
YES
Done
YES
NO
FINISH
Initialize trees
B
Best play made
YES
A
NO
Add key card to tree
NO
Finesse required
YES
FINESSE
Finesse again
YES
NO
Any cards left
NO
B
YES
PLAY

FIGURE 4

to be used only to recreate these playing hands when required. The number of cards held by declarer is determined and the number of missing honors is counted. A determination is made as to whether a finesse is required or the cards should be played consecutively. This decision is made on the basis of what honor is missing, how many cards in the suit are held by declarer, and the probability of a finesse being the play most likely to capture the missing card. Here, heuristics such as the "eight ever, nine never" rule, come into play in directing the play of the cards to one of two major sections of the program, Procedures PLAY or FINESSE.

The procedure PLAY is designed to play the cards, starting with the highest ranking card in declarer's hands, without regard to any missing key cards. PLAY is called whenever the initial program analysis determines that the probability of the key cards falling to this sort of strategy exceeds the probability of falling to a finesse. It is also called from Procedure FINESSE whenever all key cards have been played by the opposition.

PLAY accomplishes its mission by continually searching the declarer's hands to locate the highest ranking card in the suit, or an equivalent card (the Queen is equal to the Ace if the Ace, King, Queen are held by partners). The correct lead is made by determining whether the high card, or its equivalent, can be led directly, and which hand declarer wishes to be in upon completion of the play.

The strategy decided upon will play the cards in a manner requiring the fewest transportation plays per suit. For example, Finesse determines

which hand has the most cards and, whenever possible, that hand is given the lead before his partner becomes void in the suit.

When all cards have been played from the North/South hands, the program calls BETPLAY to analyze the completed play of the suit. Before looking at this portion of the program, a thorough study should be made of the Procedure FINESSE which is the heart of the finessing play. Upon entering FINESSE, the program determines the highest ranking honor missing from the declarer's hands and goes to the appropriate decision tree to find the correct card to play.

The first time a finessing hand is played, the tree will be only one level deep and will contain just the header cell and its associated FPLAY cell containing the value of the card to be played. In this case the tree traversal is limited to going to the FPLAY cell to find the finessing card. The card is then located in the North/South hands. Should it be missing, an adjustment is made in the designation of the card to be played so that the next lower ranking card held by declarer becomes the new finessing card. Both of the declarer hands are inspected for distribution of cards to be used in finessing and a decision is made as to how the finesse should be attempted. For example, if the King is missing and the other honors are distributed as shown in either of the hands below, the finesse can be made in only one way, although the exact play of the cards will be different.

North Leader

N A	N A 10
S Q J 10	S Q J

In either case, the finesse must be made through the Ace as it is recognized as the only card of a higher rank than the King, and transportation cards must be used to get to the South hand for finessing purposes. Then, assuming that the Queen is successfully finessed, the program infers that the King is located in West's hand and that it must use transportation cards to return to the South hand. Successive finesses will be attempted until the missing honor appears, declarer runs out of finessing cards, or loses his transportation capability.

It should be noted here that the assumption has been made that sufficient transportation cards will be available to allow the play of a suit to proceed as envisioned by Finesse. The routine TRANSPO is called whenever such transportation is required. It changes the lead from one hand to another and prints out the statement "Lead Changes Hands".

Whenever the missing honor appears, and when there is only one level to the decision tree, the program infers that there are no more key cards to worry about and goes to the PLAY block for completion of the suit play. If the tree level is greater than one, however, there is more than one missing card to be finessed, and control reverts to the beginning of FINESSE for another iteration.

As each complete play of four cards is made, a count is made of the key cards played by the opponents. When this count equals the number of key cards to be found, all required finesses have been taken. The routine PLAY is entered and, if there are any cards left, they are played

as required by that section of the program. When all declarer's cards have been played, Procedure BETPLAY, the initial part of the analysis of the hand, is called.

BETPLAY determines the maximum number of tricks that might be taken by declarer by counting the cards in each declarer hand and assuming that the maximum tricks possible equals the number of cards in the longer hand. If that number of tricks was actually made, BETPLAY will infer that the optimum play has been made, make a statement to that effect and terminate the play of that hand. If a lesser number of tricks was taken and it appears as if a better play might have been made, the analysis of the play continues. Simple rules are applied to determine whether it is possible to take more tricks than were made in the previous play. For example, if North has four cards and South three, then four tricks are the most that can be won. If the Ace is missing, or if the King is not held by declarer and is not finessable because it sits behind the Ace, then only three tricks can be made. If the previous play took those three tricks, the inference will be drawn that the best possible play has been made, and the play of the hand will terminate. Otherwise, the routine BPLAY will be called. This is the portion of the program in which decision trees are built and manipulated.

BPLAY immediately checks the value of all opponents' cards, with the exception of the Ace, that won tricks. The value of the highest ranking card in this category is then checked to see if it has already been used as a key card. If so, the next such card in the array is located and checked until a new key card is discovered. When the new key card is

found, the next lower ranking card is assigned as its finessing card and a new level added to the tree.

As each new key card is identified and added to the tree, Procedure CHUP is used to compare its value with the value of the card immediately above it. If the last card added is found to be of a higher rank than its predecessor, the tree positions of the two cards are interchanged, and this procedure is repeated until all cards occupy their proper place in the tree, with the higher ranking cards at the top.

If no new key cards are found, the program infers that there are no other important cards to finesse and that the optimum play has been made. The message "All Key Cards Have Been Tested" is printed out and the program looks for another hand to play.

The play and replay of a suit may also terminate when the newest play proves to be bad. Finesse determines this by comparing the number of tricks won with the number taken by the previous play. If the last play takes the same number, or more tricks than the preceding play, the program assumes the method of play to be good and leaves the information stored in the tree. If fewer tricks are taken, the line of play is considered bad and that play is removed from the decision tree. The last play remaining in the tree is considered the optimum play.

VII. CONCLUSIONS

The program Finesse does learn. It develops a line of play by successively finessing for newly discovered key cards, as long as each new play does at least as well as the previous play. Then, when the apparent best play is found, it is stored in a decision tree in memory for future use. The next time the same situation is presented for a solution, the hand is played in the same way. In a subsequent play of what appears to be an identical hand, the opponents' card distribution may cause the pre-planned play to take fewer tricks than anticipated. If so, Finesse will erase the previous play and try to find a better one. Thus Finesse is learning to finesse in much the same way that a human player would. It starts by learning a few basic bridge rules and adds to these through experience.

The rules developed could be improved upon by expanding Finesse to include more information and give the declarer a better idea of what adversities may face him. Information which might be provided includes the bidding sequence and a listing of cards already played from other suits (to show declarer something about the distribution of the entire hand). With such information added, the probability of finding the optimum play in a specific situation would increase. But this increase in information will require an equivalent increase in the size and complexity of the decision trees. If they become too detailed, the plays will become too specific, and the result may be specialization rather than generalization.

As it is, Finesse provides a good general method of play for any suit of seven cards or more. For example, any hand with the King, Jack, 9 missing will be played in the same general way, rather than in the two or three more specialized ways it might have been played if additional information were known. Special rules for each specific situation should not have to be developed. However, some additional information, particularly in the area of knowledge of the bid, would be helpful in determining which opponent to finesse through. Such an improvement should help significantly to close the gap between what Finesse does now and what it would have to do as a part of a complete bridge playing program.

By storing in the decision trees only information concerning the missing key cards, rather than those held by the declarer, the size of the decision trees was reduced, saving storage requirements and data manipulation time. This principle of working with the complement of required information may prove a valuable technique which can be used in other programming problems.

APPENDIX A

This section provides a chronological listing of major Begin Blocks and Procedures found in Finesse with a brief description of their purposes and means of accomplishment.

PSTART: This block is the entry point to Finesse. After each complete play of a suit, control of the program returns to PSTART for a determination of whether another hand will be played or the program will terminate. Subsequent action is decided by the setting of variable QUEST to a value of zero, one or two.

READIN: The purpose of this block is to provide an entry point to all program input.

LOWCARD1, LOWCARD2: These two routines find the lowest ranking cards in HAND1 and HAND2 respectively by searching the HAND1 and HAND2 arrays backwards for the first bit equal to 1. Upon termination, they return the values of variables LCARD1 and LCARD2 to the calling section of the program.

CHHAND: This is the routine called to change the lead from one declarer hand to the other when the trick has been won by a card from HAND2. When this happens, HAND1 and HAND2 exchange card values as do OHAND1 and OHAND2 and the lead then changes to the hand now called HAND1.

TWIN: Twin is designed to count the number of tricks won by the declarer hands. The variable TRICKCT returns the new trick count

value. If the trick has been won by either the East or West hand, TWIN calls Procedure OWIN.

OWIN: OWIN identifies and stores key cards in the array OHCARD. If East's nine wins a trick, for example, the bit OHCARD (6), the array position for the nine card, is set to one. OWIN then returns control to TWIN.

FOLCH: Folch is a routine called whenever the FOLLOW card is to be played. It checks the present value of FOLLOW and, by looking at the card that has been played from hand OHAND1, determines whether a change must be made in the FOLLOW value.

The play of the FOLLOW card has been pre-programmed for one of these two standard plays:

1. If the LEAD card is other than the lowest card in the HAND1 hand, that is, it is either the highest card in the hand or a selected finessing card, HAND2 is programmed to play its lowest card unless the analysis by FOLCH dictates a different play. This will occur when the OHAND1 card is higher ranking than the card led. Then, if HAND2 can beat the opponents' play, FOLCH will change the FOLLOW value to play the card from HAND2 immediately higher than that played by the first opponent.

2. If LCARD1 is to be led from HAND1, HAND2 is programmed to play either its highest card or a designated finessing card. This will be done unless FOLCH determines that OHAND1 has played a card higher in value than the programmed FOLLOW card. If so, a higher card will

be played if available. If such a card is not held, LCARD2 is played.

OPLAY1, OPLAY2: These two routines are called when the play of a card is required by the East/West hands. OPLAY1 tells opponent one to input the value of a card from his hand, and OPLAY2 does the same for OHAND2. The cards played are removed from the opponents' hands.

ONEHMISS: If only one honor is determined to be missing from the combined declarer hands, this procedure applies a set of simple rules to decide whether control should pass to the PLAY or to the FINESSE sections of the program.

PLAY: That section of Finesse which plays the declarer's cards in numerical order. It is called when there is no requirement to finesse for key cards. A detailed description of this program block and its internal blocks and procedures has been included in the Program Description section.

FINESSE: This is the section of the program which determines the cards to be used for finessing purposes and how the finesses will be made. A detailed description of FINESSE and its internal procedures and blocks has been included in the Program Description section.

TRANSPO: This section of coding is called whenever the program determines that the lead is located in the wrong hand and that a play using transportation cards is required to get the lead in the correct hand. TRANSPO does this through an artificial play of the cards designed to require CHHAND to change the cards held by all four players. The statement "LEAD CHANGES HANDS" is printed out.

BETPLAY: BETPLAY is a procedure designed to recreate the original hands for all four players, to count the number of cards in each of declarer's hands, and determine the maximum number of tricks which might possibly be made. If required, it also directs control of the play to MISS1 or MISS2.

MISS1, MISS2: By checking the suit distribution against a few distributional rules, a quick determination may be made within these program blocks concerning the maximum number of tricks that can be made. See the Program Description for a more detailed account of these program sections.

BPLAY: This procedure looks for new key cards by searching the array OHCARD. If one is found, BPLAY adds it and its associated finessing card to the decision tree and switches the program back to the FINESSE procedure so that a new line of play may be tried. BPLAY also removes plays from the decision trees when the analysis of a play shows that it takes fewer tricks than its predecessor. See the Program Description section for a more detailed discussion of BPLAY.

CHUP: CHUP is a procedure which is called after each new level has been added to the decision tree. CHUP will rearrange the tree levels if a lower level cell is found to contain a higher ranking card than its immediate predecessor in the tree.

FINISH: FINISH is a begin block used as an exit from Finesse. When there are no more hands to be played and the value of two has been entered for the variable QUEST, program control passes to FINISH, the

message "Finish" is printed, and the program terminates.

LOPUT, FOPUT: These two procedures take the array values of the LEAD and FOLLOW cards respectively and convert them to their actual card values ranging from fourteen for the Ace to two for the two. The conversion is made by searching the VTREE cells for the correct array value, then substituting for it the value found in the associated DESIG cell.

IN1, IN2: These procedures are the opposite of LOPUT and FOPUT. Their purpose is to take the OH1 and OH2 input (the cards played by opponents 1 and 2) and convert these card values to array numbers.

```
BRIDGE: PROC OPTIONS (MAIN):
/* FINESSE - A BRIDGE PROGRAM TO DEVELOP FINESSING RULES */
  DCL QUEST FIXED:
/* IF QUEST=0 THEN TREES ARE INITIALIZED, IF QUEST=1
CONTROL PASSES TO THE READIN OF DATA, IF QUEST=2 THE
PROGRAM TERMINATES */
  PSTART: BEGIN:
  GET DATA (QUEST):
  IF QUEST=1 THEN DO:
     GO TO READIN:
     END:
  ELSE IF QUEST=2 THEN GO TO FINISH:
  ELSE:
  END PSTART:
  DCL (A,B,C,D,E,F,G,H,O,P,Q,R,S,T,U,V,W,X,Y,Z) FIXED:
  DCL CDA PTR, CDK PTR, CDQ PTR, CDJ PTR:
  DCL CD PTR:
  DCL (CD1, CD2, CD3, CD4, CD5) PTR:
  DCL (SH,HMISS#,HMISS1) FIXED:
/*MCARDTREE CELLS ARE HEADER CELLS */
  DCL 1 MCARDTREE BASED(PC),
       2 MC1 FIXED,
       2 M1 PTR,
       2 M2 PTR,
       2 M3 PTR,
       2 M4 PTR,
       2 M6 PTR:
/* N3 CELLS CONTAIN DESIGNATED KEY CARDS */
  DCL 1 N3 BASED(P4),
       2 MC4 FIXED,
       2 M7 PTR,
       2 PL PTR,
       2 BP PTR:
/* FPLAY CELLS CONTAIN CARDS TO BE USED IN FINESSING
PLAY */
  DCL 1 FPLAY BASED(P2),
       2 NEXTC FIXED,
       2 EM PTR:
/* THIS SECTION INITIALIZES ALL HEADER CELLS */
  ALLOCATE MCARDTREE SET (PC):
     CDA=PC:
     CD=CDA:
     CDA->MC1=1:
  ALLOCATE FPLAY SET (P2):
     CD=P2:
     CDA->M6=CD:
     CD->NEXTC=2:
     CD->EM=NULL:
```

```
      CDA->M1=NULL;
      CDA->M2=NULL;
      CDA->M3=NULL;
      CDA->M4=NULL;
   ALLOCATE MCARDTREE SET (PC);
      CDK=PC;
      CD=CDK;
      CDK->MC1=2;
   ALLOCATE FPLAY SET (P2);
      CD=P2;
      CDK->M6=CD;
      CD->NEXTC=3;
      CD->EM=NULL;
      CDK->M1=NULL;
      CDK->M2=NULL;
      CDK->M3=NULL;
      CDK->M4=NULL;
   ALLOCATE MCARDTREE SET (PC);
      CDQ=PC;
      CD=CDQ;
      CDQ->MC1=3;
   ALLOCATE FPLAY SET (P2);
      CD=P2;
      CDQ->M6=CD;
      CD->NEXTC=4;
      CD->EM=NULL;
      CDQ->M1=NULL;
      CDQ->M2=NULL;
      CDQ->M3=NULL;
      CDQ->M4=NULL;
   ALLOCATE MCARDTREE SET (PC);
      CDJ=PC;
      CD=CDJ;
      CDJ->MC1=5;
   ALLOCATE FPLAY SET (P2);
      CD=P2;
      CDJ->M6=CD;
      CD->NEXTC=1;
      CD->EM=NULL;
      CDJ->M1=NULL;
      CDJ->M2=NULL;
      CDJ->M3=NULL;
      CDJ->M4=NULL;
/* THIS SECTION BUILDS MEMORY TO ASSOCIATE CARD ARRAY
VALUES WITH ACTUAL CARD VALUES */
   DCL (LEED,NLEED) PTR;
   DCL 1 VTREE BASED (PP),
       2 NME FIXED,
```

```
     2 DLEED PTR,
     2 CON PTR;
DCL  1 DESIG BASED (PF),
     2 NEWNME FIXED,
     2 DNLEED PTR;
ALLOCATE VTREE SET (PP);
   CD=PP;
   LEED=PP;
   LEED->NME=1;
ALLOCATE DESIG SET (PF);
   CD1=PF;
   NLEED=PF;
   NLEED->NEWNME=14;
   LEED->CON=NLEED;
DO X=2 TO 13;
   T=15-X;
   ALLOCATE VTREE SET (PP);
      LEED->DLEED=PP;
      LEED=PP;
      LEED->NME=X;
   ALLOCATE DESIG SET (PF);
      NLEED->DNLEED=PF;
      NLEED=PF;
      NLEED->NEWNME=T;
      LEED->CON=NLEED;
END;
   LEED->DLEED=NULL;
   NLEED->DNLEED=NULL;
   LEED=CD;
   NLEED=CD1;
   PP=LEED;
   PF=NLEED;
DCL OHCARD(13) BIT (1);
DCL (NORTH(13), SOUTH(13),HAND1(13),HAND2(13)) BIT (1);
DCL (WEST(13), EAST(13), OHAND1(13), OHAND2(13)) BIT (1);
DCL (LCD, LCD1, TEMP, TRICKCT, LEAD, FOLLOW) FIXED;
LEVEL=1;
NH=1; SH=2; LEADER=NH;
DCL (LCARD1,LCARD2) FIXED;
DCL WINNER FIXED;
DCL (OH1,OH2) FIXED;
DCL (CPLAYED(13), TEMPHAND(13)) BIT (1);
DCL VALUE FIXED;
DCL NCARD FIXED;
DCL HMISS FIXED;
DCL TEMPCARD(13) BIT (1);
DCL TEMPN FIXED;
DCL (LLEAD,FFOLLOW) FIXED;
```

```
  DCL LTEMP FIXED;
  READIN: BEGIN;
/* HANDS TO BE PLAYED ARE READ IN */
  GET LIST (NORTH,SOUTH,EAST,WEST);
     GET LIST (CPLAYED);
     HAND1=CPLAYED; HAND2=CPLAYED;
     TEMPHAND=CPLAYED;
     OHCARD=CPLAYED;
     TEMPCARD=OHCARD;
     TRICKCT=0;
     S=0;
     NCARD=0;
     TEMP=0;
  END READIN;
  IF LEADER=NH THEN DO;
     HAND1=NORTH; HAND2=SOUTH; OHAND1=EAST; OHAND2=WEST; END;
  ELSE DO; HAND1=SOUTH; HAND2=NORTH; OHAND1=WEST; OHAND2=EAST; END;
  CARDS=0;
  DO H=1 TO 13;
     IF HAND1(H)=1|HAND2(H)=1 THEN
        CARDS=CARDS+1;
  END;
  HMISS#=0;
  DO G=1 TO 5;
     IF HAND1(G)=0&HAND2(G)=0 THEN HMISS#=HMISS#+1; END;
  LOWCARD1: PROC;
/* THIS ROUTINE FINDS THE LOWEST CARD IN HAND1 */
  Y=13;
  LOOP: BEGIN;
     IF HAND1(Y)=1 THEN LCARD1=Y;
     ELSE IF Y>1 THEN DO;
        Y=Y-1; GO TO LOOP; END;
        ELSE DO; LCARD1=0; RETURN; END;
  END LOOP;
  END LOWCARD1;
  LOWCARD2: PROC;
/* THIS ROUTINE FINDS THE LOWEST CARD IN HAND2 */
  Y=13;
  LOOP: BEGIN;
     IF HAND2(Y)=1 THEN LCARD2=Y;
     ELSE IF Y>1 THEN DO;
        Y=Y-1; GO TO LOOP; END;
        ELSE DO; LCARD2=0; RETURN; END;
  END LOOP;
  END LOWCARD2;
  CHHAND: PROC;
/* THIS ROUTINE CHANGES THE LEAD BETWEEN NORTH/SOUTH
HANDS WHENEVER THE PREVIOUS TRICK HAS BEEN WON BY HAND2 */
```

```
      IF FOLLOW=0 THEN RETURN;
      ELSE;
      IF FOLLOW<LEAD THEN IF (OH1>FOLLOW|OH1=0)&(OH2>FOLLOW|OH2=0) THEN
         WINNER=2;
      ELSE DO; WINNER=1; RETURN;
      END;
      ELSE RETURN;
      IF WINNER=2 THEN DO;
         TEMPHAND=HAND1;
         HAND1=HAND2;
         HAND2=TEMPHAND;
         Z=A; A=B; B=Z;
         TEMPHAND=OHAND1;
         OHAND1=OHAND2;
         OHAND2=TEMPHAND;
         DO J=1 TO 13;
            TEMPHAND(J)=0; END;
      END;
   END CHHAND;
   TWIN: PROC;
/* THIS ROUTINE COUNTS TRICKS WON BY NORTH/SOUTH */
      IF OH1=0&OH2=0 THEN DO;
         TRICKCT=TRICKCT+1; PUT DATA (TRICKCT); RETURN; END;
      ELSE;
      IF FOLLOW>0 THEN IF LEAD>FOLLOW THEN
         IF (FOLLOW<OH1|OH1=0)&(FOLLOW<OH2|OH2=0) THEN
         TRICKCT=TRICKCT+1;
         ELSE CALL OWIN;
      ELSE IF OH1>0 THEN IF LEAD<OH1 THEN
         IF OH2>0 THEN IF LEAD<OH2 THEN TRICKCT=TRICKCT+1;
         ELSE CALL OWIN;
         ELSE TRICKCT=TRICKCT+1;
         ELSE CALL OWIN;
         ELSE IF OH2>0 THEN IF LEAD<OH2 THEN TRICKCT=TRICKCT+1;
         ELSE CALL OWIN;
         ELSE TRICKCT=TRICKCT+1;
      ELSE IF (LEAD<OH1|OH1=0)&(LEAD<OH2|OH2=0) THEN
         TRICKCT=TRICKCT+1;
      ELSE CALL OWIN;
      PUT DATA (TRICKCT);
   END TWIN;
   OWIN: PROC;
/* THIS ROUTINE IDENTIFIES KEY CARDS AND STORES THEM FOR
REPLAY PURPOSES */
      IF OH1>0 THEN
      IF OH1<LEAD&(OH1<FOLLOW|FOLLOW=0) THEN DO;
         LTEMP=LEAD;
         LEAD=OH1;
```

```
         CALL LOPUT;
         OH1=LEAD;
         LEAD=LTEMP;
         PUT LIST ('NEW KEY CARD IS'||OH1);
         CALL IN1;
         OHCARD(OH1)='1'B;
         RETURN;
         END;
      ELSE;
      ELSE;
      IF OH2>0 THEN
      IF OH2<LEAD&(OH2<FOLLOW|FOLLOW=0) THEN DO;
         LTEMP=LEAD;
         LEAD=OH2;
         CALL LOPUT;
         OH2=LEAD;
         LEAD=LTEMP;
         PUT LIST ('NEW KEY CARD IS'||OH2);
         CALL IN2;
         OHCARD(OH2)='1'B;
         END;
      ELSE;
      ELSE;
   END OWIN;
   FOLCH: PROC;
/* THIS ROUTINE DETERMINES WHETHER THE PLANNED VALUE OF
THE FOLLOW CARD MUST BE CHANGED TO SATISFY OPPONENT 1'S
PLAY */
      IF LEAD>OH1 THEN DO;
         IF OH1¬=1 THEN DO;
         W=OH1-1;
         IF HAND2(W)=1 THEN DO;
            DO WHILE (HAND2(W)=1);
               W=W-1; END;
            U=W+1;
            FOLLOW=U; END;
         ELSE DO; DO WHILE (HAND2(W)=0);
            W=W-1; END;
         IF W>0 THEN FOLLOW=W;
         ELSE RETURN; END;
         END;
         ELSE RETURN;
      END;
      ELSE RETURN;
   END FOLCH;
   OPLAY1: PROC;
/* OPPONENT 1'S CARD IS PLAYED BY THIS ROUTINE */
      PUT LIST ('OPPONENT 1 PLAYS');
```

```
      GET DATA (OH1);
      CALL IN1;
      OHAND1(OH1)=0;
   END OPLAY1;
   OPLAY2: PROC;
/* OPPONENT 2'S CARD IS PLAYED BY THIS ROUTINE */
      PUT LIST ('OPPONENT 2 PLAYS');
      GET DATA (OH2);
      CALL IN2;
      OHAND2(OH2)=0;
   END OPLAY2;
   LOPUT: PROC;
/* THIS ROUTINE CHANGES THE ARRAY VALUE OF THE LEAD CARD
TO THE ACTUAL CARD VALUE */
      LPUT: BEGIN;
      U=LEED->NME; W=NLEED->NEWNME;
      IF U=LEAD THEN LEAD=W;
      ELSE DO;
         LEED=LEED->DLEED;
         NLEED=NLEED->DNLEED;
         GO TO LPUT;
      END;
      LEED=PP;
      NLEED=PF;
      END LPUT;
   END LOPUT;
   FOPUT: PROC;
/* THIS ROUTINE CHANGES THE ARRAY VALUE OF THE FOLLOW CARD
TO THE ACTUAL CARD VALUE */
      FPUT: BEGIN;
      IF FOLLOW>0 THEN DO;
      U=LEED->NME; W=NLEED->NEWNME;
      IF U=FOLLOW THEN FOLLOW=W;
      ELSE DO;
         LEED=LEED->DLEED;
         NLEED=NLEED->DNLEED;
         GO TO FPUT;
      END;
      END;
      ELSE;
      END FPUT;
      LEED=PP;
      NLEED=PF;
   END FOPUT;
   IN1: PROC;
/* THIS ROUTINE CHANGES THE ACTUAL VALUE OF EACH OH1 CARD
TO ITS ARRAY VALUE */
      INONE: BEGIN;
```

```
         IF OH1>0 THEN DO:
         U=LEED->NME: W=NLEED->NEWNME:
         IF OH1=W THEN OH1=U:
         ELSE DO:
            LEED=LEED->DLEED:
            NLEED=NLEED->DNLEED:
            GO TO INONE:
         END:
         END: ELSE:
         LEED=PP:
         NLEED=PF:
      END INONE:
   END IN1:
   IN2: PROC:
/* THIS ROUTINE CHANGES THE ACTUAL VALUE OF EACH OH2 CARD
TO ITS ARRAY VALUE */
      INTWO: BEGIN:
         IF OH2>0 THEN DO:
         U=LEED->NME: W=NLEED->NEWNME:
         IF OH2=W THEN OH2=U:
         ELSE DO:
            LEED=LEED->DLEED:
            NLEED=NLEED->DNLEED:
            GO TO INTWO:
         END:
         END: ELSE:
         LEED=PP:
         NLEED=PF:
      END INTWO:
   END IN2:
   IF QUEST=1 THEN DO:
      NCARD=TEMPN:
      DO Z=1 TO 13:
         IF HAND1(Z)=0&HAND2(Z)=0 THEN TEMPCARD(Z)='1'B:
      END:
      END:
   ELSE:
   IF CARDS>6 THEN IF HMISS#<2 THEN CALL ONEHMISS:
      ELSE IF HMISS#=2 THEN
         IF HAND1(1)=0&HAND2(1)=0&HAND1(2)=0&HAND2(2)=0
            THEN CALL ONEHMISS:
         ELSE CALL FINESSE:
      ELSE CALL FINESSE:
   ELSE:
   ONEHMISS: PROC:
/* THIS ROUTINE SELECTS THE CORRECT LINE OF PLAY FOR A
HAND WITH ONLY ONE HONOR MISSING */
   G=1:
```

```
   HMISS1=0;
   IF HAND1(G)=0&HAND2(G)=0 THEN HMISS1=1;
   ELSE DO WHILE (HAND1(G)=1|HAND2(G)=1);
      G=G+1; HMISS1=G; END;
   IF HMISS1=1 THEN CALL PLAY;
   ELSE IF HMISS1=2 THEN IF CARDS>10 THEN CALL PLAY;
            ELSE CALL FINESSE;
      ELSE IF HMISS1=3 THEN IF CARDS>8 THEN CALL PLAY;
         ELSE CALL FINESSE;
         ELSE CALL PLAY;
   END ONEHMISS;
   CALL BETPLAY;
   PLAY: PROC RECURSIVE;
/* THIS ROUTINE PLAYS DECLARER'S CARDS IN ORDER FROM
HIGHEST TO LOWEST. IT IS USED WHEN KEY CARDS ARE NOT A
CONSIDERATION */
   I=0;
   EQUALCARDS: BEGIN;
/* EQUALCARDS LOCATES ALL HONORS IN THE NORTH/SOUTH HANDS
AND INITIALIZES THE CORRECT PLAY */
   P=0;
   DO J=1 TO 13;
      IF HAND1(J)=1|HAND2(J)=1 THEN P=1; END;
   IF P=0 THEN GO TO FINISH;
      A=0; B=0;
      DO N=1 TO 13;
         IF HAND1(N)=1 THEN A=A+1; END;
      DO N=1 TO 13;
         IF HAND2(N)=1 THEN B=B+1; END;
      DO N=1 TO 5;
         CALL LOWCARD1; CALL LOWCARD2;
         IF HAND1(N)=0 THEN CALL CH2;
         ELSE IF A>B THEN DO;
            X=N;
            L=N+1;
               DO WHILE (HAND1(L)=1);
                  X=L; L=L+1;
               END;
            IF HAND2(X+1)=1 THEN DO;
               Q=X; K=X+1;
               DO WHILE (HAND2(K)=1);
                  Q=K; K=K+1;
               END;
               LEAD=LCARD1;
               HAND1(LCARD1)=0;
               CALL OPLAY1;
               FOLLOW=Q;
               CALL FOLCH;
```

```
                HAND2(FOLLOW)=0;
                CALL OPLAY2;
                CALL TWIN;
                CALL CHHAND;
                LLEAD=LEAD; FFOLLOW=FOLLOW;
                CALL LOPUT;
                CALL FOPUT;
                PUT DATA (LEAD,FOLLOW);
                A=A-1; B=B-1;
            END;
            ELSE DO; LEAD=X;
                HAND1(X)=0;
                CALL OPLAY1;
                CALL LOWCARD2;
                FOLLOW=LCARD2;
                CALL FOLCH;
                HAND2(FOLLOW)=0;
                CALL OPLAY2;
                CALL TWIN;
                CALL CHHAND;
                A=A-1; B=B-1;
                LLEAD=LEAD; FFOLLOW=FOLLOW;
                CALL LOPUT;
                CALL FOPUT;
                PUT DATA (LEAD,FOLLOW);
                END;
        END;
        ELSE CALL CH3;
        END;
        CALL HIGHC;
    CH2: PROC;
/* CALLED BY EQUALCARDS WHENEVER AN HONOR TO BE PLAYED IS
FOUND IN HAND2 */
        IF HAND2(N)=0 THEN RETURN;
        ELSE IF A>B THEN DO;
            LEAD=LCARD1;
            HAND1(LCARD1)=0;
            CALL OPLAY1;
            FOLLOW=N;
            CALL FOLCH;
            HAND2(FOLLOW)=0;
            CALL OPLAY2;
            CALL TWIN;
            CALL CHHAND;
            LLEAD=LEAD; FFOLLOW=FOLLOW;
            CALL LOPUT;
            CALL FOPUT;
            PUT DATA (LEAD,FOLLOW);
```

```
         GO TO EQUALCARDS; END;
      ELSE CALL HIGHC;
   END CH2;
   CH3: PROC;
/* CALLED BY EQUALCARDS WHEN DISTRIBUTION DICTATES THAT
HONORS BE PLAYED FROM SHORT-SUITED HAND FIRST */
      LEAD=N;
      HAND1(N)=0;
      CALL OPLAY1;
      FOLLOW=LCARD2;
      CALL FOLCH;
      HAND2(FOLLOW)=0;
      CALL OPLAY2;
      CALL TWIN;
      CALL CHHAND;
      A=A-1; B=B-1;
      LLEAD=LEAD; FFOLLOW=FOLLOW;
      CALL LOPUT;
      CALL FOPUT;
      PUT DATA (LEAD,FOLLOW);
   END CH3;
   END EQUALCARDS;
   HIGHC: PROC RECURSIVE;
/* PLAYS HIGHEST CARD IN DECLARER'S HANDS WHEN CALLED
FROM EQUALCARDS */
   HIGH: BEGIN;
      P=0;
      DO J=1 TO 13;
         IF HAND1(J)=1|HAND2(J)=1 THEN P=1; END;
      IF P=0 THEN GO TO EHIGH;
      CALL LOWCARD1; CALL LOWCARD2;
      IF I<13 THEN I=I+1;
   IF HAND1(I)=1 THEN DO;
      LEAD=I;
      HAND1(I)=0;
      CALL OPLAY1;
      CALL LOWCARD2;
      FOLLOW=LCARD2;
      HAND2(FOLLOW)=0;
      LLEAD=LEAD; FFOLLOW=FOLLOW;
      CALL OPLAY2;
      CALL TWIN;
      CALL CHHAND;
      A=A-1; B=B-1;
      LLEAD=LEAD; FFOLLOW=FOLLOW;
      CALL LOPUT;
      CALL FOPUT;
      PUT DATA (LEAD,FOLLOW);
```

```
      GO TO HIGH;
      END;
   ELSE IF HAND2(I)=1 THEN DO;
      X=I; L=I+1;
         DO WHILE (HAND2(L)=1);
            X=L; L=L+1;
            END;
         IF HAND1(X+1)=1 THEN DO;
            K=X+1; Q=K;
            DO WHILE (HAND1(K)=1);
               Q=K; K=K+1;
            END;
         LEAD=Q;
         HAND1(Q)=0;
         CALL OPLAY1;
         FOLLOW=LCARD2;
         CALL FOLCH;
         HAND2(FOLLOW)=0;
         CALL OPLAY2;
         A=A-1; B=B-1;
         CALL TWIN;
         CALL CHHAND;
         LLEAD=LEAD; FFOLLOW=FOLLOW;
         CALL LOPUT;
         CALL FOPUT;
         PUT DATA (LEAD,FOLLOW);
         GO TO HIGH;
      END;
      ELSE DO;
         CALL LOWCARD1;
         IF LCARD1=0 THEN DO;
            CALL TRANSPO;
            LEAD=I;
            HAND1(I)=0;
            CALL OPLAY1;
            FOLLOW=LCARD2;
            CALL OPLAY2;
         END;
         ELSE DO; LEAD=LCARD1;
            HAND1(LCARD1)=0;
            CALL OPLAY1;
            FOLLOW=I;
            CALL FOLCH;
         HAND2(FOLLOW)=0;
            CALL OPLAY2;
      END;
      CALL TWIN;
      CALL CHHAND;
```

```
      LLEAD=LEAD; FFOLLOW=FOLLOW:
      CALL LOPUT;
      CALL FOPUT;
      PUT DATA (LEAD,FOLLOW);
      A=A-1; B=B-1;
      GO TO HIGH;
      END;
   END;
   ELSE GO TO EQUALCARDS;
   EHIGH: BEGIN;
   END EHIGH;
   END HIGH;
   END HIGHC:
   END PLAY;
   FINESSE: PROC RECURSIVE;
/* THIS ROUTINE PLAYS ALL CARDS WHILE IN A FINESSING
SITUATION. WHEN NO NEED IS LEFT TO FINESSE, THIS ROUTINE
CALLS PLAY FOR THE PLAY OF ALL REMAINING CARDS */
   PUT LIST ('FINESSE');
   A=1; B=0;
   F=0;
   CHVALUE: PROC;
/* THIS PROCEDURE ADJUSTS THE VALUE OF THE FINESSING CARD
WHEN THE CARD DESIRED IS NOT HELD BY DECLARER */
      PV=VALUE+1;
      DO WHILE (HAND1(PV)=0&HAND2(PV)=0);
         PV=PV+1; END;
      VALUE=PV;
   END CHVALUE;
   TRICKCT=0;
   D=0;
   FIN1: BEGIN;
/* THIS BLOCK SELECTS THE CORRECT DECISION TREE */
    CALL LOWCARD1: CALL LOWCARD2;
      IF LCARD1=0 THEN CALL TRANSPO;
      ELSE;
   DO I=1 TO 5;
      IF HAND1(I)=0&HAND2(I)=0 THEN
         IF I=1 THEN DO; CD=CDA; GO TO FIN2; END;
         ELSE IF I=2 THEN DO; CD=CDK; GO TO FIN2; END;
         ELSE IF I=3 THEN DO; CD=CDQ; GO TO FIN2 ; END;
         ELSE IF I=4 THEN DO; CD=CDJ; GO TO FIN2; END;
         ELSE IF I=5 THEN GO TO FIN10;
         ELSE GO TO FINISH;
      END;
   FIN10: BEGIN;
/* THIS BLOCK FINESSES FOR THE TEN IF REQUIRED. NO TEN TREE
HAS BEEN INITIALIZED */
```

```
      CALL LOWCARD1; CALL LOWCARD2;
      IF LCARD1=0 THEN CALL TRANSPO;
      ELSE;
      L=6;
         IF HAND1(L)=1 THEN DO;
            LEAD=L;
            HAND1(L)=0;
            CALL OPLAY1;
            FOLLOW=LCARD2;
            CALL FOLCH;
            HAND2(FOLLOW)=0;
            CALL OPLAY2;
            CALL TWIN;
            CALL CHHAND; END;
         ELSE IF HAND2(L)=1 THEN DO;
            CALL TRANSPO;
            LEAD=L;
            HAND1(L)=0;
            CALL OPLAY1;
            CALL LOWCARD2;
            FOLLOW=LCARD2;
            CALL FOLCH;
            HAND2(FOLLOW)=0;
            CALL OPLAY2;
            CALL TWIN;
            CALL CHHAND;
         END;
      ELSE DO; CALL PLAY; RETURN; END;
   END FIN10;
/* FOLLOWING CODE CHECKS TO SEE IF TEN WAS PLAYED */
      IF LLEAD<FFOLLOW THEN
         IF OH1<LLEAD|OH2<LLEAD THEN DO;
            CALL PLAY; CALL BETPLAY; GO TO FINISH; END;
         ELSE IF OH1<FFOLLOW|OH2<FFOLLOW THEN DO;
            CALL PLAY; CALL BETPLAY; GO TO FINISH;
         END;
         ELSE GO TO FIN1;
   END FIN1;
   FIN2: BEGIN;
/* THIS BLOCK SEARCHES DECISION TREE FOR CORRECT PLAY
AND FINESSES AS OFTEN AS REQUIRED */
      S=S+1;
      P=0;
      DO L=1 TO 13;
         IF HAND1(L)=1|HAND2(L)=1 THEN P=1; END;
      IF P=0 THEN DO; CALL BETPLAY; GO TO FINISH; END;
      ELSE;
      CALL LOWCARD1; CALL LOWCARD2;
```

```
      IF LCARD1=0 THEN CALL TRANSPO;
      ELSE;
      CD1=CD;
      IF QUEST=1 THEN DO;
         HMISS=CD->MC1;
         NUM=HMISS+1;
/* CODE SELECTS THE CORRECT BRANCH OF THE DECISION TREE */
         DO Z=NUM TO 13;
            IF TEMPCARD(Z)='1'B THEN IF Z=HMISS+1 THEN DO;
               TEMPCARD=TEMPHAND;
               CD=CD->M1;
               GO TO DOWNTREE1;
               END;
            ELSE IF Z=HMISS+2 THEN DO;
               TEMPCARD=TEMPHAND;
               CD=CD->M2;
               GO TO DOWNTREE2;
            END;
            ELSE IF Z=HMISS+3 THEN DO;
               TEMPCARD=TEMPHAND;
               CD=CD->M3;
               GO TO DOWNTREE3;
            END;
            ELSE IF Z=HMISS+4 THEN DO;
               TEMPCARD=TEMPHAND;
               CD=CD->M4;
               GO TO DOWNTREE4;
            END;
            ELSE DO; CD=CD->M1;
               GO TO DOWNTREE1;
            END;
            ELSE;
         END;
      END;
      ELSE;
/* FOLLOWING SECTION TRAVERSES DECISION TREES */
      IF CD->M1¬=NULL THEN DO;
         LEVEL=2;
         CD=CD->M1;
            DOWNTREE1: BEGIN;
         IF CD1->M1=NULL THEN GO TO OUT4;
         HMISS=CD->MC4;
         IF LEVEL=2 THEN IF HAND1(HMISS)=1|HAND2(HMISS)=1
            THEN GO TO OUT1;
         TEMPCARD(HMISS)='1'B;
         IF HAND1(HMISS)=0&HAND2(HMISS)=0 THEN DO;
            IF CD->M7=NULL THEN DO;
               CD2=CD;
```

```
            CD=CD->PL;
            VALUE=CD->NEXTC;
            IF HAND1(VALUE)=0&HAND2(VALUE)=0 THEN DO;
               CD=CD2->BP;
               IF LEVEL=2 THEN CD=CD->M6;
               ELSE CD=CD->PL;
               VALUE=CD->NEXTC;
            END;
            ELSE;
         END;
         ELSE DO; CD=CD->M7;
         LEVEL=LEVEL+1;
         X=CD->MC4;
         TEMPCARD(X)='1'B;
      GO TO DOWNTREE1; END;
      END;
         END DOWNTREE1;
END;
ELSE DO;
OUT1: BEGIN;
CD=CD1;
END OUT1;
IF CD->M2¬=NULL THEN DO;
   LEVEL=2;
   CD=CD->M2;
      DOWNTREE2: BEGIN;
   IF CD1->M2=NULL THEN GO TO OUT4;
   HMISS=CD->MC4;
   IF LEVEL=2 THEN IF HAND1(HMISS)=1|HAND2(HMISS)=1
      THEN GO TO OUT2;
   TEMPCARD(HMISS)='1'B;
   IF HAND1(HMISS)=0&HAND2(HMISS)=0 THEN DO;
      IF CD->M7=NULL THEN DO;
         CD2=CD;
         CD=CD->PL;
         VALUE=CD->NEXTC;
         IF HAND1(VALUE)=0&HAND2(VALUE)=0 THEN DO;
            CD=CD2->BP;
            IF LEVEL=2 THEN CD=CD->M6;
            ELSE CD=CD->PL;
            VALUE=CD->NEXTC;
         END;
         ELSE;
      END;
      ELSE DO; CD=CD->M7;
      LEVEL=LEVEL+1;
      X=CD->MC4;
      TEMPCARD(X)='1'B;
```

```
   GO TO DOWNTREE2; END;
   END;
      END DOWNTREE2;
END;
ELSE DO;
OUT2: BEGIN;
CD=CD1;
END OUT2;
IF CD->M3¬=NULL THEN DO;
   LEVEL=2;
   CD=CD->M3;
      DOWNTREE3: BEGIN;
   IF CD1->M3=NULL THEN GO TO OUT4;
   HMISS=CD->MC4;
   IF LEVEL=2 THEN IF HAND1(HMISS)=1|HAND2(HMISS)=1
      THEN GO TO OUT3;
   TEMPCARD(HMISS)='1'B;
   IF HAND1(HMISS)=0&HAND2(HMISS)=0 THEN DO;
      IF CD->M7=NULL THEN DO;
         CD2=CD;
         CD=CD->PL;
         VALUE=CD->NEXTC;
         IF HAND1(VALUE)=0&HAND2(VALUE)=0 THEN DO;
            CD=CD2->BP;
            IF LEVEL=2 THEN CD=CD->M6;
            ELSE CD=CD->PL;
            VALUE=CD->NEXTC;
         END;
         ELSE;
      END;
      ELSE DO; CD=CD->M7;
      LEVEL=LEVEL+1;
      X=CD->MC4;
      TEMPCARD(X)='1'B;
   GO TO DOWNTREE3; END;
   END;
      END DOWNTREE3;
END;
ELSE DO;
OUT3: BEGIN;
CD=CD1;
END OUT3;
IF CD->M4¬=NULL THEN DO;
   LEVEL=2;
   CD=CD->M4;
      DOWNTREE4: BEGIN;
   IF CD1->M4=NULL THEN GO TO OUT4;
HMISS=CD->MC4;
```

```
            IF LEVEL=2 THEN IF HAND1(HMISS)=1|HAND2(HMISS)=1
               THEN GO TO OUT4;
            TEMPCARD(HMISS)='1'B;
            IF HAND1(HMISS)=0&HAND2(HMISS)=0 THEN DO:
               IF CD->M7=NULL THEN DO:
                  CD2=CD:
                  CD=CD->PL:
                  VALUE=CD->NEXTC:
                  IF HAND1(VALUE)=0&HAND2(VALUE)=0 THEN DO:
                     CD=CD2->BP:
                     IF LEVEL=2 THEN CD=CD->M6:
                     ELSE CD=CD->PL:
                     VALUE=CD->NEXTC:
                  END:
                  ELSE:
               END:
               ELSE DO: CD=CD->M7:
               LEVEL=LEVEL+1:
               X=CD->MC4:
               TEMPCARD(X)='1'B:
            GO TO DOWNTREE4: END:
            END:
               END DOWNTREE4:
         END:
         ELSE DO:
         OUT4: BEGIN:
         CD=CD1:
         END OUT4:
            LEVEL=1:
            CD=CD->M6:
            VALUE=CD->NEXTC:
         END:
      END: END: END:
   /* THIS SECTION FINESSES FOR AN UNLOCATED KEY CARD */
         IF F=0 THEN DO:
         IF HAND1(VALUE)=1 THEN DO: V=VALUE-2:
            IF V>0 THEN IF HAND2(V)=0 THEN DO:
               X=1:
               IF X<V THEN DO WHILE (HAND2(X)=0):
                  X=X+1:
                  IF X=V THEN GO TO OUTLOOP:
                  ELSE: END:
               ELSE GO TO OUTLOOP:
               IF HAND1(VALUE)=0&HAND2(VALUE)=0
                  THEN CALL CHVALUE:
               ELSE:
               LEAD=VALUE:
               HAND1(VALUE)=0:
```

```
      CALL OPLAY1:
      FOLLOW=LCARD2:
      CALL FOLCH:
      HAND2(FOLLOW)=0:
      CALL OPLAY2:
      CALL TWIN:
      CALL CHHAND:
      LLEAD=LEAD: FFOLLOW=FOLLOW:
      CALL LOPUT:
      CALL FOPUT:
      PUT DATA (LEAD,FOLLOW):
      END:
   ELSE DO:
      LEAD=VALUE:
      HAND1(VALUE)=0:
      CALL OPLAY1:
      FOLLOW=LCARD2:
      CALL FOLCH:
      HAND2(FOLLOW)=0:
      CALL OPLAY2:
      CALL TWIN:
      CALL CHHAND:
      LLEAD=LEAD: FFOLLOW=FOLLOW:
      CALL LOPUT:
      CALL FOPUT:
      PUT DATA (LEAD,FOLLOW):
      END:
   ELSE DO:
      OUTLOOP: BEGIN:
      END OUTLOOP:
      CALL TRANSPO:
      IF S>1 THEN GO TO HCARD:
      LEAD=LCARD1:
      HAND1(LCARD1)=0:
      CALL OPLAY1:
      IF HAND1(VALUE)=0&HAND2(VALUE)=0
         THEN CALL CHVALUE:
      ELSE:
      FOLLOW=VALUE:
      CALL FOLCH:
      HAND2(FOLLOW)=0:
      CALL OPLAY2:
      IF FOLLOW=VALUE&OH2>FOLLOW THEN F=1:
      ELSE:
      IF OH1<=NCARD|OH2<=NCARD THEN F=0:
      ELSE:
      CALL TWIN:
      CALL CHHAND:
```

```
      LLEAD=LEAD; FFOLLOW=FOLLOW;
      CALL LOPUT;
      CALL FOPUT;
      PUT DATA (LEAD,FOLLOW);
      IF F=1 THEN DO; CALL TRANSPO;
         GO TO F1;
      END;
   END;
END;
ELSE DO;
   IF S>1 THEN DO;
   HCARD: BEGIN;
      CALL LOWCARD1; CALL LOWCARD2;
      W=HMISS; U=W;
      DO WHILE (HAND1(W)=0);
         W=W+1; END;
      DO WHILE (HAND2(U)=0);
         U=U+1; END;
      IF W<U THEN DO;
         LEAD=W;
         HAND1(W)=0;
         CALL OPLAY1;
         FOLLOW=LCARD2;
         CALL FOLCH;
         HAND2(FOLLOW)=0;
         CALL OPLAY2;
         END;
      ELSE DO; LEAD=LCARD1;
         HAND1(LCARD1)=0;
         CALL OPLAY1;
         FOLLOW=U;
         CALL FOLCH;
         HAND2(FOLLOW)=0;
         CALL OPLAY2; END;
   END HCARD;
   END;
   ELSE DO; LEAD=LCARD1;
      HAND1(LCARD1)=0;
      CALL OPLAY1;
      IF HAND1(VALUE)=0&HAND2(VALUE)=0
         THEN CALL CHVALUE;
      ELSE;
      FOLLOW=VALUE;
      IF VALUE>0H1 THEN CALL FOLCH;
      ELSE FOLLOW=VALUE;
      HAND2(FOLLOW)=0;
      CALL OPLAY2;
   END;
```

```
            IF FOLLOW=VALUE&OH2>FOLLOW THEN F=1:
            ELSE:
            IF OH1<=NCARD|OH2<=NCARD THEN F=0:
            ELSE:
            CALL TWIN:
            CALL CHHAND:
            LLEAD=LEAD: FFOLLOW=FOLLOW:
            CALL LOPUT:
            CALL FOPUT:
            PUT DATA (LEAD,FOLLOW):
            IF F=1 THEN DO: CALL TRANSPO:
               GO TO F1:
            END:
         END:
         END:
         ELSE DO:
         F1: BEGIN:
/* WHEN A KEY CARD HAS BEEN LOCATED, THIS BLOCK INSURES
THAT THE FINESSE IS MADE THROUGH THE CORRECT OPPONENT */
            IF HAND1(VALUE)=0&HAND2(VALUE)=0 THEN CALL CHVALUE:
            ELSE:
            IF HAND1(VALUE)=1 THEN DO:
               LEAD=VALUE:
               HAND1(VALUE)=0:
               CALL OPLAY1:
               CALL LOWCARD2:
               FOLLOW=LCARD2:
               CALL FOLCH:
               HAND2(FOLLOW)=0:
               CALL OPLAY2:
               CALL TWIN:
               CALL CHHAND:
               LLEAD=LEAD: FFOLLOW=FOLLOW:
               CALL LOPUT:
               CALL FOPUT:
               PUT DATA (LEAD,FOLLOW):
               END:
            ELSE DO:
               CALL LOWCARD1:
               LEAD=LCARD1:
               HAND1(LCARD1)=0:
               CALL OPLAY1:
               FOLLOW=VALUE:
               IF FOLLOW>OH1 THEN CALL FOLCH:
               ELSE:
               HAND2(FOLLOW)=0:
               CALL OPLAY2:
               CALL TWIN:
```

```
            CALL CHHAND;
            LLEAD=LEAD; FFOLLOW=FOLLOW;
            CALL LOPUT;
            CALL FOPUT;
            PUT DATA (LEAD,FOLLOW);
         END;
         IF OH1<VALUE|OH2<VALUE THEN F=2;
         ELSE;
         IF F=1 THEN GO TO F1;
         ELSE IF F=2 THEN DO; P=0;
      DO J=1 TO 13;
         IF HAND1(J)=1|HAND2(J)=1 THEN P=1; END;
      IF P=1 THEN DO; CALL PLAY; CALL BETPLAY; END;
      ELSE;
         END;
      END F1;
      END;
      CD=CD1->M6;
   IF LEVEL=1 THEN DO;
/* THIS SECTION DETERMINES WHETHER THE HONOR IN THE
HEADER CELL HAS BEEN PLAYED. ONLY ONE CARD IS SOUGHT */
      IF LLEAD<FFOLLOW THEN
         IF OH1<LLEAD|OH2<LLEAD THEN D=D+1;
         ELSE;
      ELSE IF OH1<LLEAD THEN DO; D=D+1;
         IF OH2<FFOLLOW THEN D=D+1;
         ELSE;
      IF D=1 THEN
         IF CD1->M1=NULL&CD1->M2=NULL&CD1->M3=NULL
            &CD1->M4=NULL THEN DO;
               CALL PLAY; CALL BETPLAY; END;
         ELSE;
         ELSE IF D=2 THEN DO;
            CALL PLAY; CALL BETPLAY; END;
         ELSE IF D=1 THEN IF VALUE=CD->NEXTC THEN DO;
            CALL PLAY; CALL BETPLAY; END;
         ELSE DO; CD=CD1; GO TO FIN2; END;
         ELSE IF VALUE=CD->NEXTC THEN DO;
            DO WHILE (HAND1(VALUE+1)=0&HAND2(VALUE+1)=0);
               VALUE=VALUE+1; END;
            IF HAND1(VALUE)=1 THEN DO;
               LEAD=VALUE;
               HAND1(VALUE)=0;
               CALL OPLAY1;
               FOLLOW=LCARD2;
               CALL FOLCH;
               HAND2(FOLLOW)=0;
               CALL OPLAY2; END;
```

```
                ELSE DO; LEAD=LCARD1;
                    HAND1(LCARD1)=0;
                    CALL OPLAY1;
                    FOLLOW=VALUE;
                    IF VALUE>OH1 THEN CALL FOLCH;
                    ELSE FOLLOW=VALUE;
                    HAND2(FOLLOW)=0;
                    CALL OPLAY2; END;
                    CALL TWIN;
                    CALL CHHAND;
                    LLEAD=LEAD; FFOLLOW=FOLLOW;
                    CALL LOPUT;
                    CALL FOPUT;
                    PUT DATA (LEAD,FOLLOW);
                END;
                ELSE DO; CD=CD1; GO TO FIN2; END;
            END;
            ELSE DO; CD=CD1; GO TO FIN2; END;
    END;
/* THIS SECTION DETERMINES WHETHER ALL KEY CARDS HAVE
BEEN PLAYED */
    ELSE IF OH1=NCARD|OH2=NCARD THEN DO; D=D+1;
        IF OH1=NCARD&OH2<NCARD THEN D=D+1;
        ELSE; END;
        ELSE IF OH1<NCARD|OH2<NCARD THEN D=D+1;
        ELSE IF OH1<NCARD&OH2<NCARD THEN D=D+2;
        IF D<2 THEN DO; CD=CD1; GO TO FIN2; END;
        ELSE DO; CALL PLAY; RETURN; END;
    END FIN2;
    END FINESSE;
    TRANSPO: PROC;
/* THIS ROUTINE PROVIDES TRANSPORTATION TO THE OPPOSITE
HAND */
        LEAD=2; FOLLOW=1;
        OH1=3; OH2=3;
        CALL CHHAND;
        CALL LOWCARD1; CALL LOWCARD2;
        PUT LIST ('LEAD CHANGES HANDS');
    END TRANSPO;
    BETPLAY: PROC RECURSIVE;
/* THIS ROUTINE INITIATES THE REPLAY OF THE HAND TO TRY
FOR A BETTER PLAY */
    R=0;
    PUT DATA (TRICKCT);
    IF LEADER=NH THEN DO;
        HAND1=NORTH; HAND2=SOUTH; OHAND1=EAST; OHAND2=WEST; END;
    ELSE DO; HAND1=SOUTH; HAND2=NORTH; OHAND1=WEST; OHAND2=EAST;
    END;
```

```
   K=0; L=0;
   DO J=1 TO 13;
      IF NORTH(J)=1 THEN K=K+1;
      ELSE IF SOUTH(J)=1 THEN L=L+1; END;
   IF K>L THEN M=K;
   ELSE M=L; /* M=MAX TRICKS POSSIBLE */
   PUT LIST ('MAXIMUM TRICKS POSSIBLE='||M);
   IF M=TRICKCT THEN DO;
      PUT LIST ('PLAY MADE MAXIMUM TRICKS');
      GO TO PSTART; END;
   IF HMISS#=1 THEN DO;
      CALL MISS1; IF R=1 THEN GO TO PSTART; END;
   ELSE DO; CALL MISS2; IF R=1 THEN GO TO PSTART; END;
   END BETPLAY;
   MISS1: PROC RECURSIVE;
/* IF ONE HONOR IS MISSING, THIS ROUTINE DECIDES WHETHER
ADDITIONAL ANALYSIS IS REQUIRED TO FIND THE OPTIMUM PLAY */
      IF HMISS1=1 THEN
         IF TRICKCT=M-1 THEN DO;
            PUT LIST ('PLAY MADE MAXIMUM TRICKS');
            R=1;
            GO TO PSTART; END;
         ELSE CALL BPLAY;
      ELSE IF HMISS1=2 THEN DO;
         IF HAND1(1)=1 THEN DO;
            IF OHAND2(2)=1 THEN IF TRICKCT=M-1 THEN DO;
               PUT LIST ('PLAY MADE MAXIMUM TRICKS');
               R=1;
               GO TO PSTART; END;
            ELSE CALL BPLAY;
      END;
      ELSE IF HAND2(1)=1 THEN DO;
         IF OHAND2(2)=1 THEN IF TRICKCT=M-1 THEN DO;
            PUT LIST ('PLAY MADE MAXIMUM TRICKS');
            R=1;
            GO TO PSTART; END;
         ELSE CALL BPLAY; END;
   END MISS1;
   MISS2: PROC RECURSIVE;
/* IF TWO HONORS ARE MISSING, THIS ROUTINE DECIDES
WHETHER ADDITIONAL ANALYSIS IS REQUIRED TO FIND THE
OPTIMUM PLAY */
      IF HAND1(1)=0&HAND2(1)=0 THEN IF HAND1(2)=0&HAND2(2)=0 THEN
         IF TRICKCT=M-2 THEN DO;
            PUT LIST ('PLAY MADE MAXIMUM TRICKS');
            R=1;
            GO TO PSTART; END;
         ELSE CALL BPLAY;
```

```
      ELSE IF HAND1(3)=0&HAND2(3)=0 THEN DO:
         IF HAND1(2)=1 THEN IF OHAND1(3)=1 THEN
         IF TRICKCT=M-2 THEN DO:
            PUT LIST ('PLAY MADE MAXIMUM TRICKS');
            R=1;
            GO TO PSTART; END;
         ELSE CALL BPLAY;
         ELSE CALL BPLAY;
      ELSE IF HAND2(2)=1 THEN IF OHAND2(3)=1 THEN
         IF TRICKCT=M-2 THEN DO;
            PUT LIST ('PLAY MADE MAXIMUM TRICKS');
            R=1;
            GO TO PSTART; END;
         ELSE CALL BPLAY;
         ELSE CALL BPLAY;
         ELSE CALL BPLAY;
      END;
      ELSE CALL BPLAY;
      ELSE CALL BPLAY;
   END MISS2;
   ECARDS: PROC;
/* THIS ROUTINE FINDS THE LOWEST EQUIVALENT CARD FOR
WHICH THE PROGRAM MUST FINESSE */
      X=I; L=I+1;
      IF OHAND1(X)=1 THEN DO WHILE (OHAND1(L)=1);
         X=L; L=L+1; END;
      ELSE DO; X=I; L=I+1;
         DO WHILE (OHAND2(L)=1);
            X=L; L=L+1; END;
      END;
      NCARD=X;
      L=X+1;
      IF OHAND1(L)=1 THEN DO WHILE (OHAND1(L)=1);
            X=L; L=L+1; END;
      ELSE IF OHAND2(L)=1 THEN DO WHILE (OHAND2(L)=1);
            X=L; L=L+1; END;
      ELSE;
      NCARD=X;
   END ECARDS;
   BPLAY: PROC RECURSIVE;
/* THE ROUTINE TO BUILD AND MANIPULATE DECISION TREES */
   DCL SEARCH PTR;
   DCL CRD FIXED;
   A=1; B=0;
   NCARD=0;
   DO I=2 TO 13;
      CALL LOWCARD1;
      IF OHCARD(I)=1 THEN DO:
```

```
            IF TEMPCARD(I)¬=1 THEN DO:
            TEMPCARD(I)='1'B:
            CALL ECARDS: LDC=NCARD+1:
            IF HAND1(LDC)=1 THEN LEAD=LDC:
            ELSE IF HAND1(LDC+1)=1 THEN DO:
               DO WHILE (HAND1(LDC+1)=1):
               LDC1=LDC+1:
               LDC=LDC+1:
               END:
               LEAD=LDC1:
               END:
            ELSE DO: LEAD=LCARD1:
            FOLLOW=LDC:
            END:
            GO TO DONELOOP:
            END:
         ELSE:
         END:
            ELSE:
            END:
         DONELOOP: BEGIN:
         END DONELOOP:
            TEMP=TRICKCT:
            OHCARD=TEMPHAND:
            IF NCARD=0 THEN DO:
               PUT LIST ('ALL KEY CARDS HAVE BEEN TESTED'):
               GO TO PSTART:
            END:
            ELSE TEMPN=NCARD:
/* THE CORRECT DECISION TREE IS FOUND */
  IF HAND1(1)=0&HAND2(1)=0 THEN CD=CDA:
  ELSE IF HAND1(2)=0&HAND2(2)=0 THEN CD=CDK:
  ELSE IF HAND1(3)=0&HAND2(3)=0 THEN CD=CDQ:
  ELSE CD=CDJ:
  CRD=CD->MC1:
  IF LEVEL=1 THEN DO:
     CD1=CD:
/*  TREES ARE BUILT */
     ALLOCATE N3 SET (P4):
     IF I<=5 THEN
     IF I=CRD+1 THEN CD->M1=P4:
     ELSE IF I=CRD+2 THEN CD->M2=P4:
     ELSE IF I=CRD+3 THEN CD->M3=P4:
     ELSE IF I=CRD+4 THEN CD->M4=P4:
     ELSE CD->M1=P4:
     ELSE CD->M1=P4:
     CD3=CD->M6:
     CD=P4:
```

```
    CD->BP=CD1;
    CD2=P4;
    CD5=CD2;
    CD2->MC4=I;
    ALLOCATE FPLAY SET (P2);
    CD=P2;
    CD2->PL=CD;
    CD->NEXTC=LDC;
    CD2->M7=NULL;
    CD4=CD;
    CD->EM=CD3;
    LEVEL=2;
    NEWVAL=CD->NEXTC;
END;
ELSE IF CD2->M7¬=NULL THEN DO;
    IF HAND1(CRD+1)=0&HAND2(CRD+1)=0 THEN SEARCH=CD1->M1;
    ELSE IF HAND1(CRD+2)=0&HAND2(CRD+2)=0 THEN SEARCH=CD1->M2;
    ELSE IF HAND1(CRD+3)=0&HAND2(CRD+3)=0 THEN SEARCH=CD1->M3;
    ELSE IF HAND1(CRD+4)=0&HAND2(CRD+4)=0 THEN SEARCH=CD1->M4;
    ELSE SEARCH=CD1->M1;
    IF SEARCH->MC4¬=I THEN DO WHILE (SEARCH->M7¬=NULL);
    SEARCH=SEARCH->M7;
    IF SEARCH->MC4=I THEN GO TO DONE;
END; END;
ELSE DO;
    ALLOCATE N3 SET (P4);
    CD=P4;
    CD2->M7=CD;
    CD->MC4=I;
    CD->BP=CD2;
    CD5=CD2;
    CD2=CD;
    ALLOCATE FPLAY SET (P2);
    CD=P2;
    CD2->PL=CD;
    CD->EM=CD4;
    CD->NEXTC=LDC;
    CD2->M7=NULL;
    CD4=CD;
    LEVEL=LEVEL+1;
    CALL CHUP;
    NEWVAL=CD->NEXTC;
    END;
DONE: BEGIN;
END DONE;
    S=0;
    CALL FINESSE;
    PUT LIST ('PREVIOUS TRICK COUNT='||ITEMP);
```

```
      IF TRICKCT<TEMP THEN DO:
/*  IF THE LAST PLAY WAS BAD, IT IS DISCARDED */
      IF LEVEL>=2 THEN DO:
         CD4=CD4->EM:
         FREE CD->FPLAY:
         CD=CD2:
         CD2=CD2->BP:
         FREE CD->N3:
         CD=CD4:
         CD2->M7=NULL:
         LEVEL=LEVEL-1:
         END:
      ELSE DO:
         CD4=CD4->EM:
         FREE CD->FPLAY:
         CD=CD2:
         CD2=CD2->BP:
         IF CD->MC4=CRD+1 THEN CD2->M1=NULL:
         ELSE IF CD->MC4=CRD+2 THEN CD2->M2=NULL:
         ELSE IF CD->MC4=CRD+3 THEN CD2->M3=NULL:
         ELSE CD2->M4=NULL:
         FREE CD->N3:
         CD=CD4:
         LEVEL=1:
         END:
         PUT LIST ('NEW VALUE NOT REQUIRED: FREE CELL'):
      GO TO PSTART:
         END:
      ELSE CALL BETPLAY:
   END BPLAY:
   CHUP: PROC:
/*  THIS ROUTINE ARRANGES ALL LEVELS OF THE TREES IN
NUMERICAL ORDER */
      IF CD2->MC4<CD5->MC4 THEN
         IF LEVEL>2 THEN DO: LEVEL1=LEVEL:
            UPTREE: BEGIN:
               CD5->M7=CD2->M7:
               CD2->M7=CD5:
               CD2->BP=CD5->BP:
               CD5->BP=CD2:
               CD4=CD->EM:
               CD->EM=CD4->EM:
               CD4->EM=CD:
               CD=CD4:
               CD5=CD2->BP:
               CD5->M7=CD2:
               LEVEL1=LEVEL1-1:
            END UPTREE: END:
```

```
      ELSE IF LEVEL=2 THEN RETURN;
         ELSE IF LEVEL1>2 THEN IF CD2->MC4<CD5->MC4 THEN GO TO UPTREE;
            ELSE;
         ELSE IF LEVEL1=2 THEN IF CD2->MC4<CD5->MC4 THEN DO;
            CD2->BP=CD5->BP;
            CD5->BP=CD2;
            CD5->M7=CD2->M7;
            CD2->M7=CD5;
            IF CD2->MC4=CRD+1 THEN CD1->M1=CD2;
            ELSE IF CD2->MC4=CRD+2 THEN CD1->M2=CD2;
            ELSE IF CD2->MC4=CRD+3 THEN CD1->M3=CD2;
            ELSE IF CD2->MC4=CRD+4 THEN CD1->M4=CD2;
            ELSE CD1->M1=CD2;
            END;
         ELSE; ELSE; ELSE;
   END CHUP;
   FINISH: BEGIN;
      PUT LIST ('FINISH');
   END FINISH;
      STOP;
END BRIDGE;
```

LIST OF REFERENCES

1. Berlekamp, E. R., "Program for Double-Dummy Bridge Problems--A New Strategy for Mechanical Game Playing", Journal of the Association for Computing Machinery, v. 10, p. 357-364, July 1963.

2. Carley, G. L., A Program to Play Contract Bridge, M. S. Thesis, Massachusetts Institute of Technology, June 1962.

3. Throop, T. A., "The Univac Plays Bridge", COMPUTERS AND AUTOMATION, v. XI, p. 3B-5B, March 1962.

4. Riley, W., and Throop, T. A., An Interactive Bridge Playing Program, paper presented at National Gaming Council Symposium, 1969.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	LTJG Robert Bolles, USNR Department of Mathematics (Code 53) Naval Postgraduate School Monterey, California 93940	1
4.	LTCOL John H. Smith, USMC 8 Biddle Lane Monterey, California 93940	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
Finesse

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Master's Thesis; June 1970

5. AUTHOR(S) (First name, middle initial, last name)
John Henry Smith

6. REPORT DATE
June 1970

7a. TOTAL NO. OF PAGES
67

7b. NO. OF REFS
4

8a. CONTRACT OR GRANT NO.

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
Naval Postgraduate School
Monterey, California 93940

13. ABSTRACT

Finesse is a program written to investigate the role of a generalization scheme in computer learning. The finessing play in the game of bridge is developed as an example in which the computer starts with a few basic bridge rules and improves its play through self-analysis of successive plays of the same hands. After deciding that an optimum play has been found for a particular hand, Finesse makes the generalization that all hands with approximately the same distribution should be played in the same way.

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Finesse						
Bridge						
Game-playing						
Artificial intelligence						

2 NOV 72
23 OCT 73
21970

Thesis
S594 Smith
c.1 Finesse.